```
->SELECT TOP 2 * FROM Customers;
->SELECT TOP 50 PERCENT * FROM Customers;
```

SQL is a standard language for accessing and manipulating databases.

What is SQL?
SQL stands for Structured Query Language
SQL lets you access and manipulate databases
SQL is an ANSI (American National Standards Institute) standard

----

Using SQL in Your Web Site
To build a web site that shows data from a database, you will need:

An RDBMS database program (i.e. MS Access, SQL Server, MySQL)
To use a server-side scripting language, like PHP or ASP
To use SQL to get the data you want
To use HTML / CSS

RDBMS
RDBMS stands for Relational Database Management System.

RDBMS is the basis for SQL, and for all modern database systems such as MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.

The data in RDBMS is stored in database objects called tables.

A table is a collection of related data entries and it consists of columns and rows.

```
->SELECT * FROM Customers;
->SELECT CustomerName,City FROM Customers;
```

```
---------------------------------------------------
----
The SQL SELECT DISTINCT Statement
In a table, a column may contain many duplicate va
lues; and sometimes you only want to list the diff
erent (distinct) values.

The DISTINCT keyword can be used to return only dis
tinct (different) values.
->SELECT DISTINCT column_name,column_name
FROM table_name;
->SELECT DISTINCT City FROM Customers;
---------------------------------------------------
-----
The SQL WHERE Clause
The WHERE clause is used to extract only those rec
ords that fulfill a specified criterion.
->SELECT * FROM Customers
WHERE Country='Mexico';
->SELECT * FROM Customers
WHERE CustomerID=1;
---------------------------------------------------
--------------
The SQL AND & OR Operators
The AND operator displays a record if both the fir
st condition AND the second condition are true.

The OR operator displays a record if either the fi
rst condition OR the second condition is true.
->SELECT * FROM Customers
WHERE Country='Germany'
AND City='Berlin';
->SELECT * FROM Customers
WHERE City='Berlin'
OR City='München';
->SELECT * FROM Customers
WHERE Country='Germany'
AND (City='Berlin' OR City='München');
---------------------------------------------------
-----------
The SQL ORDER BY Keyword
The ORDER BY keyword is used to sort the result-set
```

```
 by one or more columns.

The ORDER BY keyword sorts the records in ascendin
g order by default. To sort the records in a desce
nding order, you can use the DESC keyword.
->SELECT * FROM Customers
ORDER BY Country;
->SELECT * FROM Customers
ORDER BY Country DESC;
->SELECT * FROM Customers
ORDER BY Country, CustomerName;
->SELECT * FROM Customers
ORDER BY Country ASC, CustomerName DESC;
--------------------------------------------------
--------
The SQL INSERT INTO Statement
The INSERT INTO statement is used to insert new rec
ords in a table.
->INSERT INTO Customers (CustomerName, ContactName,
 Address, City, PostalCode, Country)
VALUES ('Cardinal','Tom B. Erichsen','Skagen 21','S
tavanger','4006','Norway');
->INSERT INTO Customers (CustomerName, City, Countr
y)
VALUES ('Cardinal', 'Stavanger', 'Norway');
--------------------------------------------------
------------
->UPDATE Customers
SET City='Hamburg'
WHERE CustomerID=1;

UPDATE Multiple Columns
To update more than one column, use a comma as sepe
rator.

Assume we wish to update the customer "Alfreds Fut
terkiste" with a new contact person and city.

->UPDATE Customers
SET ContactName='Alfred Schmidt', City='Frankfurt'
WHERE CustomerID=1;
```

->UPDATE Customers
SET ContactName='Juan'
WHERE Country='Mexico';

NOTE:-Be careful when updating records. If we omit the WHERE clause, ALL records will be updated:

The DELETE statement is used to delete records in a table.
->DELETE FROM Customers
WHERE CustomerName='Alfreds Futterkiste' AND ContactName='Maria Anders';

Delete All Data
=It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:
->DELETE FROM table_name;

or

->DELETE * FROM table_name;

SQL Injection
SQL injection is a technique where malicious users can inject SQL commands into an SQL statement, via a web page input.

Injected SQL commands can alter SQL statement and compromise the security of a web application.

-SQL Injection Based on 1=1 is Always True
Look at the example above, one more time.

Let's say that the original purpose of the code was to create an SQL statement to select a user with a given user id.

If there is nothing to prevent a user from enterin
g "wrong" input, the user can enter some "smart" i
nput like this:

->SELECT * FROM Users WHERE UserId = 105 or 1=1;

The SQL SELECT TOP Clause
The SELECT TOP clause is used to specify the number
 of records to return.

The SELECT TOP clause can be very useful on large
tables with thousands of records. Returning a larg
e number of records can impact on performance.
->SELECT TOP 2 * FROM Customers;
->SELECT TOP 50 PERCENT * FROM Customers;

SQL LIKE Operator

=The LIKE operator is used in a WHERE clause to sea
rch for a specified pattern in a column.

The following SQL statement selects all customers w
ith a City starting with the letter "s":
->SELECT * FROM Customers
WHERE City LIKE 's%';

Tip: The "%" sign is used to define wildcards (mis
sing letters) both before and after the pattern. Y
ou will learn more about wildcards in the next cha
pter.

The following SQL statement selects all customers w
ith a City ending with the letter "s":
->SELECT * FROM Customers
WHERE City LIKE '%s';

The following SQL statement selects all customers
with a Country containing the pattern "land":
->SELECT * FROM Customers

WHERE Country LIKE '%land%';

Using the NOT keyword allows you to select records that do NOT match the pattern.

The following SQL statement selects all customers with Country NOT containing the pattern "land":
```
->SELECT * FROM Customers
WHERE Country NOT LIKE '%land%';
```

SQL Wildcards:
A wildcard character can be used to substitute for any other character(s) in a string.

The following SQL statement selects all customers w
ith a City starting with "ber":
```
->SELECT * FROM Customers
WHERE City LIKE 'ber%';
```

the following SQL statement selects all customers w
ith a City containing the pattern "es":
```
->SELECT * FROM Customers
WHERE City LIKE '%es%';
```

The following SQL statement selects all customers with a City starting with any character, followed by "erlin":
```
->SELECT * FROM Customers
WHERE City LIKE '_erlin';
```

The following SQL statement selects all customers with a City starting with "L", followed by any cha
racter, followed by "n", followed by any character
,
followed by "on":
```
->SELECT * FROM Customers
WHERE City LIKE 'L_n_on';
```

The following SQL statement selects all customers w

```
ith a City starting with "b", "s", or "p":
SELECT * FROM Customers
WHERE City LIKE '[bsp]%';

The following SQL statement selects all customers w
ith a City starting with "a", "b", or "c":
->SELECT * FROM Customers
WHERE City LIKE '[a-c]%';

The two following SQL statements selects all custo
mers with a City NOT starting with "b", "s", or "p
":
->SELECT * FROM Customers
WHERE City LIKE '[!bsp]%';

or

->SELECT * FROM Customers
WHERE City NOT LIKE '[bsp]%';
-----------------------------------------------------
------------------
The IN Operator
The IN operator allows you to specify multiple valu
es in a WHERE clause.

The following SQL statement selects all customers w
ith a City of "Paris" or "London":
->SELECT * FROM Customers
WHERE City IN ('Paris','London');

-----------------------------------------------------
-------------------
The BETWEEN operator is used to select values withi
n a range.

The SQL BETWEEN Operator
The BETWEEN operator selects values within a range
. The values can be numbers, text, or dates.
->SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20;

To display the products outside the range of the pr
```

evious example, use NOT BETWEEN:
```
->SELECT * FROM Products
WHERE Price NOT BETWEEN 10 AND 20;
```

The following SQL statement selects all products w
ith a price BETWEEN 10 and 20, but products with a
 CategoryID of 1,2, or 3 should not be displayed:
```
->SELECT * FROM Products
WHERE (Price BETWEEN 10 AND 20)
AND NOT CategoryID IN (1,2,3);
```

The following SQL statement selects all products w
ith a ProductName beginning with any of the letter
 BETWEEN 'C' and 'M':
```
->SELECT * FROM Products
WHERE ProductName BETWEEN 'C' AND 'M';
```

The following SQL statement selects all products w
ith a ProductName beginning with any of the letter
 NOT BETWEEN 'C' and 'M':
```
->SELECT * FROM Products
WHERE ProductName NOT BETWEEN 'C' AND 'M';
```

The following SQL statement selects all orders wit
h an OrderDate BETWEEN '04-July-1996' and '09-July
-1996':
```
->SELECT * FROM Orders
WHERE OrderDate BETWEEN #07/04/1996# AND #07/09/199
6#;
```

Notice that the BETWEEN operator can produce differ
ent result in different databases!
In some databases, BETWEEN selects fields that are
between and excluding the test values.
In other databases, BETWEEN selects fields that ar
e between and including the test values.
And in other databases, BETWEEN selects fields bet
ween the test values, including the first test val
ue and excluding the last test value.

SQL aliases are used to temporarily rename a table or a column heading.

SQL Aliases
SQL aliases are used to give a database table, or a column in a table, a temporary name.

Basically aliases are created to make column names more readable.

Alias Example for Table Columns
The following SQL statement specifies two aliases, one for the CustomerName column and one for the ContactName column.
Tip: It requires double quotation marks or square brackets if the column name contains spaces:

```
->SELECT CustomerName AS Customer, ContactName AS [
Contact Person]
FROM Customers;
```

In the following SQL statement we combine four columns (Address, City, PostalCode, and Country) and create an alias named "Address":

```
->SELECT CustomerName, Address+', '+City+', '+Posta
lCode+', '+Country AS Address
FROM Customers;
```

To get the SQL statement above to work in MySQL use the following:

```
->SELECT CustomerName, CONCAT(Address,', ',City,',
',PostalCode,', ',Country) AS Address
FROM Customers;
```

Alias Example for Tables
The following SQL statement selects all the orders from the customer with CustomerID=4 (Around the Horn). We use the "Customers" and

"Orders" tables, and give them the table aliases of "c" and "o" respectively (Here we have used aliases to make the SQL shorter):

```
->SELECT o.OrderID, o.OrderDate, c.CustomerName
FROM Customers AS c, Orders AS o
WHERE c.CustomerName="Around the Horn" AND c.Custom
erID=o.CustomerID;
```

The same SQL statement without aliases:

```
->SELECT Orders.OrderID, Orders.OrderDate, Customer
s.CustomerName
FROM Customers, Orders
WHERE Customers.CustomerName="Around the Horn" AND
 Customers.CustomerID=Orders.CustomerID;
```

Aliases can be useful when:

There are more than one table involved in a query
Functions are used in the query
Column names are big or not very readable
Two or more columns are combined together

SQL joins are used to combine rows from two or more tables.

SQL JOIN
An SQL JOIN clause is used to combine rows from two or more tables, based on a common field between them.
The most common type of join is: SQL INNER JOIN (simple join). An SQL INNER JOIN returns all rows from multiple tables where the join condition is met.

Notice that the "CustomerID" column in the "Orders" table refers to the "CustomerID" in the "Customers" table. The relationship between the two tables above is the "CustomerID" column.

Then, if we run the following SQL statement (that c
ontains an INNER JOIN):

```
->SELECT Orders.OrderID, Customers.CustomerName, Or
ders.OrderDate
FROM Orders
INNER JOIN Customers
ON Orders.CustomerID=Customers.CustomerID;
```

Different SQL JOINs
Before we continue with examples, we will list the
 types of the different SQL JOINs you can use:

INNER JOIN: Returns all rows when there is at least
 one match in BOTH tables
LEFT JOIN: Return all rows from the left table, and
 the matched rows from the right table
RIGHT JOIN: Return all rows from the right table,
and the matched rows from the left table
FULL JOIN: Return all rows when there is a match in
 ONE of the tables

SQL INNER JOIN Keyword
The INNER JOIN keyword selects all rows from both
tables as long as there is a match between the col
umns in both tables.

```
->SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
INNER JOIN Orders
ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

Note: The INNER JOIN keyword selects all rows from
 both tables as long as there is a match between t
he columns.
If there are rows in the "Customers" table that do

 not have matches in "Orders", these customers wil
l NOT be listed.

SQL LEFT JOIN Keyword
The LEFT JOIN keyword returns all rows from the le
ft table (table1), with the matching rows in the r
ight table (table2).
 The result is NULL in the right side when there is
 no match.

SQL LEFT JOIN Syntax

```
->SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name=table2.column_name;


     OR
->SELECT column_name(s)
FROM table1
LEFT OUTER JOIN table2
ON table1.column_name=table2.column_name;
```


SQL LEFT JOIN Example
The following SQL statement will return all custome
rs, and any orders they might have:

```
->SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders
ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

Note: The LEFT JOIN keyword returns all the rows f
rom the left table (Customers), even if there are
no matches in the right table (Orders).

SQL RIGHT JOIN Keyword
The RIGHT JOIN keyword returns all rows from the r
ight table (table2), with the matching rows in the
 left table (table1).
The result is NULL in the left side when there is n
o match.

SQL RIGHT JOIN Syntax

```
->SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name=table2.column_name;

       OR
->SELECT column_name(s)
FROM table1
RIGHT OUTER JOIN table2
ON table1.column_name=table2.column_name;
```

The following SQL statement will return all employe
es, and any orders they have placed:

```
->SELECT Orders.OrderID, Employees.FirstName
FROM Orders
RIGHT JOIN Employees
ON Orders.EmployeeID=Employees.EmployeeID
ORDER BY Orders.OrderID;
```

Note: The RIGHT JOIN keyword returns all the rows
from the right table (Employees), even if there ar
e no matches in the left table (Orders).

SQL FULL OUTER JOIN Keyword
The FULL OUTER JOIN keyword returns all rows from the left table (table1) and from the right table (table2).

The FULL OUTER JOIN keyword combines the result of both LEFT and RIGHT joins.

SQL FULL OUTER JOIN Syntax:
```
->SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name=table2.column_name;
```

The following SQL statement selects all customers, and all orders:
```
->SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders
ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

Note: The FULL OUTER JOIN keyword returns all the rows from the left table (Customers), and all the rows from
the right table (Orders). If there are rows in "Cus tomers"
 that do not have matches in "Orders", or if there
 are rows in "Orders" that do not have matches in "Customers", those rows will be listed as well.

The SQL UNION Operator
The UNION operator is used to combine the result-se t of two or more SELECT statements.
Notice that each SELECT statement within the UNION must have the same number of columns.
The columns must also have similar data types. Als

o, the columns in each SELECT statement must be in the same order.

SQL UNION Syntax:-

```
->SELECT column_name(s) FROM table1
 UNION
 SELECT column_name(s) FROM table2;
```

Note: The UNION operator selects only distinct values by default. To allow duplicate values, use the ALL keyword with UNION.

```
->SELECT column_name(s) FROM table1
 UNION ALL
 SELECT column_name(s) FROM table2;
```

PS: The column names in the result-set of a UNION are usually equal to the column names in the first SELECT statement in the UNION.

SQL UNION Example:-

The following SQL statement selects all the different cities (only distinct values) from the "Customers" and the "Suppliers" tables:

```
->SELECT City FROM Customers
  UNION
  SELECT City FROM Suppliers
  ORDER BY City;
```

The following SQL statement uses UNION ALL to select all (duplicate values also) cities from the "Customers" and "Suppliers" tables:

```
->SELECT City FROM Customers
UNION ALL
SELECT City FROM Suppliers
ORDER BY City;
```

SQL UNION ALL With WHERE:-
The following SQL statement uses UNION ALL to select all (duplicate values also) German cities from the "Customers" and "Suppliers" tables:

```
->SELECT City, Country FROM Customers
WHERE Country='Germany'
UNION ALL
SELECT City, Country FROM Suppliers
WHERE Country='Germany'
ORDER BY City;
```

```
--------------------------------------------------
--------------------------------------------------
--------------------------------------------------
---
```